Рекурентни уравнения

Тодор Дуков

Защо са ни рекурентни уравнения?

Те се появяват по естествен път, когато искаме да анализираме сложността на рекурсивни алгоритми. Нека вземем за пример алгоритъма за двоично търсене:

```
1  int binary_search(int *arr, int left, int right, int val)
2  {
3      if (left > right)
4          return -1;
5
6      int mid = left + (right - left) / 2;
7
8      if (arr[mid] == val)
9          return mid;
10     else if (arr[mid] < val)
11         return binary_search(arr, mid + 1, right, val);
12     return binary_search(arr, left, mid - 1, val);
13 }
```

При подаден сортиран масив от числа `arr` с размер `n` и стойност `val`, функцията $\texttt{binary\_search}(\texttt{arr}, 0, \texttt{n} - 1, \texttt{val})$ ще върне индекс на `arr`, в който се намира `val`, ако има такъв, иначе ще върне -1. Нека помислим каква е сложността на алгоритъма. Управляващите параметри на рекурсията са `left` и `right`. Всеки път разликата между двете намалява двойно (като накрая когато $\texttt{left} = \texttt{right}$ тя ще стане отрицателна). Това означава, че в най-лошият случай сложността на алгоритъма може да се опише със следното рекурентно уравнение:

$$T(0) = 2 \text{ // заради ред 3 и 4}$$
$$T(n+1) = T(\lfloor \frac{n+1}{2} \rfloor) + 3 \text{ // заради проверките и рекурсивното извикване}$$

В този случай лесно се вижда асимптотиката на $T(n)$:

$$T(n) = T(\lfloor \frac{n}{2} \rfloor) + 1 = T(\lfloor \frac{n}{4} \rfloor) + 1 + 1 = T(\lfloor \frac{n}{8} \rfloor) + 1 + 1 + 1 = \cdots = T(0) + \underbrace{1 + \cdots + 1}_{\text{около } \log(n) \text{ пъти}} \asymp \log(n)$$

Така получаваме, че алгоритъмът има сложност $O(\log(n))$. Обаче в общият случай далеч не е толкова лесно да се намери асимптотичното поведение на дадено рекурентно уравнение. Целта ни ще бъде да развием по-богат апарат за асимптотичен анализ на рекурентните уравнения.

Начини за намиране на асимптотиката на рекурентни уравнения

Начините се разделят на два типа:

1. със решаване на уравнението
2. без решаване на уравнението

И двата начина са ценни. Първият начин ни дава формула във явен вид, което може да ни е от полза. Понякога обаче формулата във явен вид не е *"красива"*, или изобщо не може да се намери такава. Тогава идва на помощ вторият начин. Той директно ни дава някаква *"хубава"* формула, без да трябва да намираме в явен вид решение на рекурентното уравнение. Проблема е обаче, че асимптотиката понякога е малко лъжлива – алгоритъм със сложност $2^{2^{2^{1000}}}$ е асимптотично по-бавен от алгоритъм със сложност n, но практически вторият е по-бърз.

Ще разгледаме следните методи (повечето от които са разглеждани по дискретна математика):

- налучкване и доказване
- развиване (което преди малко показахме)
- методът с характеристичното уравнение
- мастър-теоремата

Нека разгледаме един пример с налучкване:

$$\begin{aligned} &T(0) = 3 \\ &T(n+1) = (n+1)T(n) - n \end{aligned}$$

Започваме да разписваме:

n	$T(n)$	$n!$
0	3	1
1	3	1
2	5	2
3	13	6
4	49	24
5	241	120
6	1441	720

Вече лесно можем да покажем с индукция, че $T(n) = 2(n!) + 1$:

- $T(0) = 3 = 2 \cdot 1 + 1 = 2 \cdot 0! + 1$
- $T(n+1) = (n+1)T(n) - n \overset{\text{(ИП)}}{=} (n+1)(2(n!)+1) - n = (n+1)(2(n!)) + n + 1 - n = 2(n+1)! + 1$

Накрая получаваме, че $T(n) \asymp n!$
Нека сега да видим как можем да използваме метода на характеристичното уравнение:

$$T(n) = 1 + \sum_{i=0}^{n-1} T(i) \text{ // функцията е добре дефинирана и за 0}$$

Рекурентното уравнение, зададено в този вид, не може да се реши с този метод. За това ще трябва да направим преобразувания:

$$\begin{aligned} &T(0) = 0 \\ &T(n+1) = 1 + \sum_{i=0}^{n} T(i) = 1 + T(n) + \underbrace{\sum_{i=0}^{n-1} T(i)}_{T(n)} = 2T(n) + 1 = \underbrace{2T(n)}_{\text{хомогенна част}} + \underbrace{1 \cdot 1^{n+1}}_{\text{нехомогенна част}} \end{aligned}$$

От хомогенната получаваме характеристичното уравнение $x - 2 = 0$ с единствен корен 2, а от нехомогенната част получаваме единствен корен 1. Така:

$$T(n) = A \cdot 2^n + B \cdot 1^n \text{ за някои константи } A, B$$

Вече няма нужда и да се намират константите – ясно е че $T(n) \asymp 2^n$. Като използваме метода на характеристичното уравнение, не е нужно да намираме накрая константите за да разберем каква е асимптотиката. Достатъчно е да вземем събираемото, която расте най-много (в случая 2^n).
Нека сега разгледаме и последният начин:

Мастър-теорема. *Нека $a \geq 1$, $b > 1$ и $f \in \mathcal{F}$. Нека $T(n) = aT(\frac{n}{b}) + f(n)$, където $\frac{n}{b}$ се интерпретира като $\lfloor \frac{n}{b} \rfloor$ или $\lceil \frac{n}{b} \rceil$. Тогава:*

1 сл. Ако $f(n) \preceq n^{\log_b(a) - \varepsilon}$ за някое $\varepsilon > 0$, то тогава $T(n) \asymp n^{log_b(a)}$

2 сл. Ако $f(n) \asymp n^{log_b(a)}$, то тогава $T(n) \asymp n^{log_b(a)} \log(n)$

3 сл. Ако са изпълнени следните условия:

1. $f(n) \succeq n^{\log_b(a) + \varepsilon}$ за някое $\varepsilon > 0$
2. съществува $0 < c < 1$, за което от някъде нататък $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$,

то тогава $T(n) \asymp f(n)$

Нека разгледаме рекурентното уравнение:

$$T(n) = 2T(\frac{n}{2}) + 1$$

Тук $a = b = 2$, и $f(n) = 1$. Също така $\log_b(a) = 1$, откъдето $f(n) = 1 \preceq n^{\log_b(a)-\varepsilon}$, за $\varepsilon \in (0, 1)$. Така по 2 сл. на мастър-теоремата получаваме, че $T(n) \asymp n$.

Задачи

Задача 1. Да се намери асимптотиката на средната сложност по време на алгоритъма за бързо сортиране т.е. на рекурентното уравнение:

$$T(n) = \frac{1}{n}\left(\sum_{i=1}^{n-1} T(i) + T(n-i)\right) + n$$

Задача 2. Да се намери асимптотиката на следните рекурентни уравнения:

$$T_1(n) = 29T_1(\frac{n}{3}) + 2\sum_{i=1}^{n}\frac{1}{i^2} \qquad T_2(n) = 29T_2(\frac{n}{3}) + 12n + \sqrt{n} \qquad T_3(n) = T_3(n-1) + \frac{n}{(n+1)(n-1)}$$

$$T_4(n) = 29T_4(\frac{n}{3}) + (\sum_{i=1}^{n}\frac{1}{i})^4 \qquad T_5(n) = 29T_5(\frac{n}{3}) + 2\sum_{i=1}^{n} i^2 \qquad T_6(n) = 29T_6(\frac{n}{3}) + n^{\sqrt{n}} + (\sqrt{n})^n$$

$$T_7(n) = T_7(\sqrt{n}) + n \qquad T_8(n) = 29T_8(\frac{n}{3}) + \binom{2n}{2} \qquad T_9(n) = 8T_9(n-1) - T_9(n-2) + 2n2^{2n} + 3n2^{3n}$$

Задача 3. Да се намери сложността по време на следния алгоритъм:

```
int alg1(int n)
{
    if (n < 2)
        return n;

    int acc = 0;

    for (int i = 0; i < n; ++i)
    {
        acc += i;
    }

    return acc + alg1(n - 1) + alg1(n - 1) + alg1(n - 2);
}
```

Ще се промени ли нещо ако връщаме $\texttt{acc} + 2 * \texttt{alg1}(\texttt{n} - 1) + \texttt{alg1}(\texttt{n} - 2)$?

Задача 4. Да се намери сложността по време на следния алгоритъм:

```
int alg2(int n)
{
    if (n < 2)
        return 2;

    int t = 0;
    t += alg2(n / 3);

    for (int i = 2; i < n; i *= 2)
    {
        t++;
    }

    t *= alg2(n / 3);

    return t;
}
```